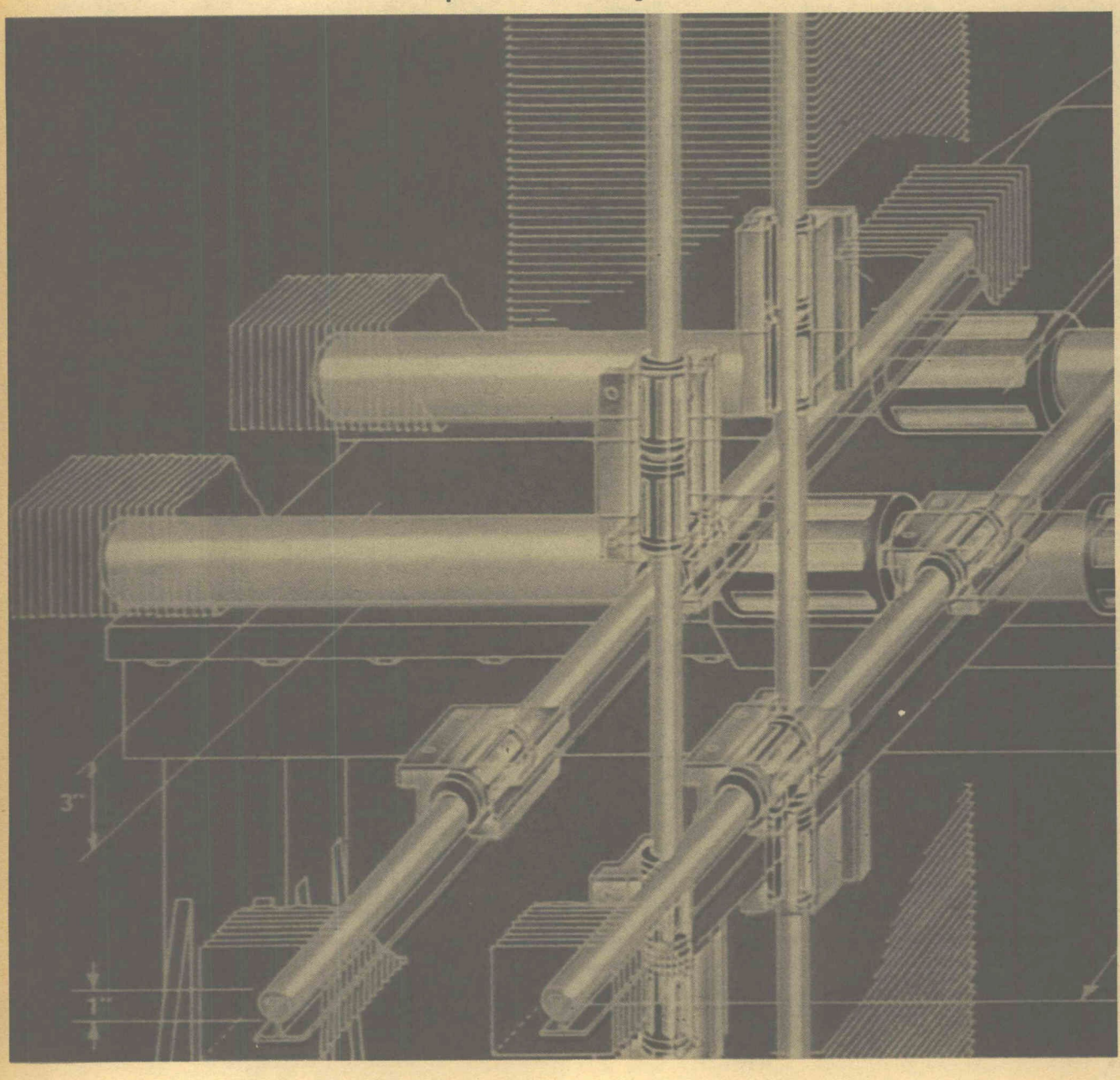
แอดวานซ์ AutoCAD

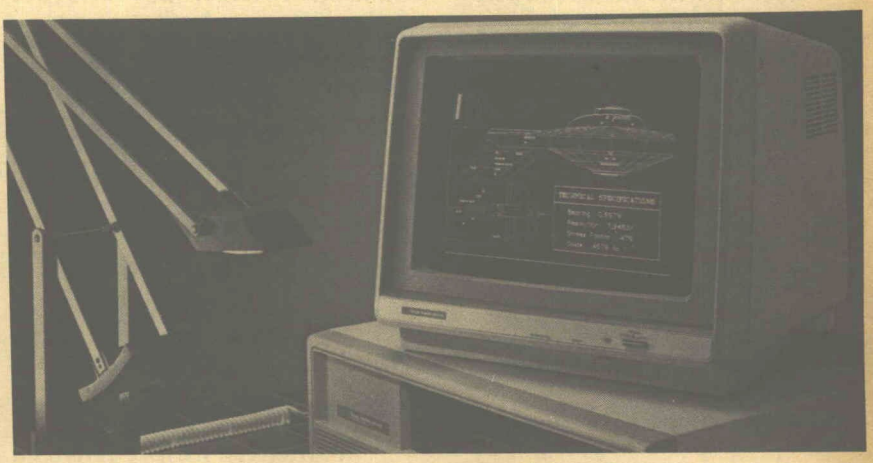
ตอน 8

สุรพล พฤกษ์ไพบูลย์



ในบทก่อน ๆ ได้กล่าวถึงเรื่องของภาษา AutoLISP เฉพาะในส่วนที่คล้ายกับภาษา LISP ไปแล้ว ในหลักการจะเห็นว่ามีส่วนที่คล้ายคลึงและใกล้เคียงกันกับ CommonLISP ซึ่งถูกจัดว่าเป็นภาษา LISP มาตรฐานเป็นอย่างมาก สำหรับในตอนต่อไปนี้จะเป็นการยกตัวอย่างการเขียนฟังก์ชันภาษา AutoLISP จากการใช้งานจริง แต่จะขอตัดคำอธิบายในส่วนของภาษา AutoLISP โดยเฉพาะออกไปเนื่องจากเห็นว่าบทความนี้มีความยาวไม่เกินกว่าที่ตั้งใจไว้แต่เดิมมาก ภาษา AutoLISP ที่เป็นส่วนของ AutoCAD โดยเฉพาะจะได้ถูกอธิบายรวมกับตัวอย่างเหล่านี้ ซึ่งจะทำให้ผู้อ่านมีความเข้าใจถึงการใช้งานได้ชัดเจนมากขึ้น ในทางตรงกันข้ามก็อาจจะทำให้เกิดความสับสนบ้าง เพราะจะไม่มีการอธิบายคำสั่งไปตามลำดับลำดาเหมือนอย่างที่เคย

ตัวอย่างย่อย ๆ ที่ได้แสดงเอาไว้ในบทนี้ จะประกอบกันเป็นตัวอย่างที่จะนำไปใช้งานได้จริงในตัวอย่างสุดท้ายคือ ml.lsp ซึ่งจะเป็นโปรแกรมที่ใช้เขียนเส้นตรงสองเส้นขนานกันซึ่งใช้เป็นเส้นผนังของงานสถาปัตยกรรม (ในรูปแปลน) บางท่านอาจจะนึกในใจว่า เอ! ฟังก์ชันทำนองนี้มีคนเขียนเอาไว้บ้างแล้ว ทั้งในหนังสือภาษาอังกฤษและภาษาไทย (ทั้งภาคธรรมดาและภาคพิสดาร) ไอ้หมอนี่เอามาให้ดูอีกทีก็ไม่เห็นจะน่าสนใจที่ตรงไหน ครับ! ก็ลองติดตามดูสักนิดว่าฟังก์ชันธรรมดา ๆ อันนี้จะมีวิธีคิดและเขียนแบบที่ไม่ธรรมดาตรงไหน จริงอยู่ ที่เราสามารถจะเขียนโปรแกรมเหล่านี้ได้โดยใช้วิธีธรรมดาสามัญตามแบบการเขียนโปรแกรมภาษาอื่น ๆ เช่น ซี หรือ ปาสคาล แต่ในเมื่อ AutoCAD เลือกที่จะมีภาษา LISP อยู่ในโปรแกรม ถึงแม้ว่าจะไม่เป็น LISP ที่มีความสามารถแก่กล้าเต็มที่



เหมือนอย่างกับ CommonLISP นักแต่ตัว AutoLISP ก็ดำรงความเป็น LISP ได้อย่างพอสมควร อันภาษา LISP นี้ก็เช่นเดียวกันกับภาษาปัญญาประดิษฐ์เช่น โปรล็อก คือมีโครงสร้างของภาษาที่เอื้ออำนวยความสะดวกให้แก่ผู้เขียนโปรแกรมขึ้นจาก concept ได้อย่างง่ายดายและสั้นได้ใจความมาก ดังนั้นการที่เราใช้ฟังก์ชันอันเป็นเฉพาะของภาษา LISP เช่น cdr, append, mapcar หรือ lambda นี้ไม่ใช่เรื่องตลกที่เราจะมองข้ามไปเพียงเพราะความยุ่งยากที่เกิดขึ้นจากความไม่คุ้นเคยหรือความที่ไม่เคยใช้ ดังจะเห็นได้ในตัวอย่างของฟังก์ชันในบทนี้ที่ได้คัดเลือกจากการใช้งานจริง และเป็นฟังก์ชันที่เขียนขึ้นโดยใช้เอกลักษณ์เฉพาะตัวของภาษา LISP ซึ่งเราก็จะได้ใช้ความรู้เกี่ยวกับภาษา LISP ที่อธิบายเอาไว้ในบทก่อน ๆ มาใช้กันอย่างเต็มที่ ใครที่อ่านบทความนี้ไปนานมาแล้ว ก็เตรียมตัวเอามาซักซ้อมความเข้าใจและนำมาใช้ได้ในบทนี้ครับ

UTILITY FUNCTION

ตัวอย่างที่ 1 ถึงตัวอย่างที่ 6 จะเป็นยูทิลิตี้ฟังก์ชัน ซึ่งเราสามารถเรียกใช้จากโปรแกรมอื่น ๆ ในทางปฏิบัติแล้วโปรแกรมเหล่านี้จะถูกรวบรวมอยู่ในไฟล์ที่ชื่อว่า ACAD. LSP (ถ้าท่านยังจำได้ผมเคยกล่าวถึง ACAD. LSP ไปแล้วว่าเป็นโปรแกรมที่ถูก load โดย Auto CAD ทุกครั้งอย่างอัตโนมัติเมื่อเราเข้าไปใน Drawing editor) และจะเป็นตัวอย่างสำหรับการเขียนโปรแกรมภาษา LISP ที่ได้อธิบายเอาไว้แล้วในตอนที่แล้ว เพราะส่วนใหญ่จะเป็นฟังก์ชันที่เขียนขึ้นโดยใช้เทคนิคพิเศษของภาษานี้โดยเฉพาะดังที่ได้กล่าวไว้แล้ว จะขออธิบายเรียงตามลำดับตั้งแต่ตัวอย่างที่หนึ่งเลยนะครับ

```
; (setqv '(<variable  name> <value> <variable name> <value>... ))
; "Setq" type of setvar system variable

(defun setqv (a)
  (if a (progn (setvar (car a) (cadr a)) (setqv (cddr a))))
)
```

ตัวอย่างที่ 1

ตัวอย่างแรกจะเป็นการเขียนฟังก์ชัน โดยการนำเอารูปแบบการใช้งานของ setq มาประยุกต์เข้าใช้กับ setvar ซึ่งเป็นคำสั่งที่ใช้กำหนดค่าตัวแปรระบบของ AutoCAD การกำหนดค่าของตัวแปรนี้ปกติแล้วจะกระทำได้ตามแบบคือ

(setvar <variable name> <value>)

(setvar "blipmode" 0)

การกำหนดค่านี้ได้จะต้องใช้ setvar ต่อการเซตค่าหนึ่งตัวเสมอ จะเห็นได้ว่าถ้าหากเราต้องการกำหนดค่าตัวแปรระบบนี้มากกว่าหนึ่งจะต้องใช้ setvar ทุกครั้งไปเช่น

(setvar "highlight" 0) (setvar "orthomode" 0)

ซึ่งจะแตกต่างอย่างมากกับฟังก์ชัน setq ที่มีรูปแบบการใช้งานดังเช่น

(setq a 1 b 2 c 3 d 4)

เหตุที่มาของฟังก์ชันนี้เป็นเพราะความขี้เกียจของผู้เขียน จึงอยากจะให้มีการเอ่ยชื่อของฟังก์ชัน setvar ขึ้นเพียงครั้งเดียวจากนั้นการเซตค่าก็ดำเนินไปเป็นคู่ ๆ ระหว่างชื่อตัวแปรและค่าของมันแบบเดียวกันกับ setq โดยที่ไม่ต้องเขียนคำว่า "setvar" และวงเล็บเปิดวงเล็บปิดซ้ำ ๆ กันหลาย ๆ หน ดังนั้นจึงเขียนฟังก์ชัน setqv ขึ้นเพื่อเลียนแบบการทำงานของ setq โดยให้มีการใช้งานดังเช่น

(setqv '("highlight" 0 "orthomode" 0 "blipmode" 1))

แต่มีข้อแม้อยู่นิดเดียวตรงที่ว่าอาร์กิวเมนต์หรือพารามิเตอร์ที่จะให้แก่ฟังก์ชัน setqv นี้จะต้องเป็นลิสต์ และถ้าพิจารณาให้ดีจะพบว่าอาร์กิวเมนต์ที่ผ่านให้ฟังก์ชันนี้มีเพียงตัวเดียวเท่านั้น ครับ! ตัวเดียวที่เป็นลิสต์เสียด้วย ดังที่เห็นในตัวอย่างข้างบนจะมี 'คือ single quote (อยู่ตรงหน้าวงเล็บเปิดอันที่สอง) ซึ่งถ้าท่านยังจำคำอธิบายในบทก่อน ๆ ได้ single quote จะมีผลทำให้สิ่งที่ตามหลังมันมาจะไม่ต้องถูกตีค่า (evaluate) นั่นคือสิ่งต่อไปจากนี้จะถูกส่งผ่านเข้าสู่ฟังก์ชันทั้งอัน และสิ่งที่อยู่ภายในวงเล็บนี้ภาษา LISP เราเรียกว่าลิสต์ ฟังก์ชัน setqv นี้เขียนขึ้นโดยใช้หลักเรียกใช้ตนเองของภาษา LISP ที่เรียกว่า Tail recursion ที่ผู้เขียนได้เคยกล่าวถึงไว้บ้างในบทก่อน ๆ แล้วนั่นเอง

ค่าของอาร์กิวเมนต์ของฟังก์ชันจะถูกทดสอบก่อนว่าเป็นค่าที่ใช้การได้คือเป็นลิสต์ จากนั้นฟังก์ชันจะดึงเอาตัวแรกของลิสต์นี้ออกมาโดยใช้ car และดึงเอาตัวที่สองของลิสต์ออกโดยใช้ cadr เพื่อให้กับฟังก์ชัน setvar ทำหน้าที่เซตค่าของตัวแปรต่อไป แล้วจึงจะผ่านส่วนที่เหลือของลิสต์ (หลังจากที่เอาสมาชิกสองตัวหน้าออกแล้ว) โดยใช้ cddr ให้กับ setq โดยการเรียกใช้ตัวเองต่อไป ซึ่งเรื่องราวจะเป็นอย่างนี้เรื่อยไปจนกว่าส่วนที่เหลือจากการเอาสมาชิกสองตัวแรกของลิสต์ออกไปแล้วเป็น nil คือไม่เป็นอะไรเลยการทำงานของฟังก์ชันในส่วนนี้จะสิ้นสุดลง และกลับคืนสู่การทำงานของฟังก์ชันชั้นบนเรื่อย ๆ จนกว่าจะถึงชั้นนอกสุดเป็นอันเสร็จสิ้นสมบูรณ์

ขอให้ลองนึกภาพหรือเขียนโปรแกรมตามแบบภาษาทั่วไป หรือใช้วิธีปกติ เช่น for หรือ while จะพบว่าการใช้วิธีเรียกตนเองนี้ทำให้สามารถที่เขียนโปรแกรมได้สั้นมาก อีกทั้งการจัดข้อมูลให้เป็นลิสต์แล้วใช้ car, cadr, cdr, cddr เพื่อดึงเอาข้อมูลมาใช้งาน รวมกันแล้วเป็นเอกลักษณ์ที่โดดเด่นเป็นอย่างมากของภาษา LISP

ตัวอย่างที่สองจะเป็นตัวอย่างการใช้ฟังก์ชัน command (ซึ่งเป็นฟังก์ชันเฉพาะของ AutoLISP) เพื่อที่จะใช้ชุดคำสั่งใน Drawing editor ของ Auto CAD เช่น LINE, ERASE, MOVE เป็นต้น เนื่องจาก color เป็นตัวแปรของ AutoCAD ที่ถูกจัดว่าเป็น Read-Only variable คือไม่สามารถที่จะกำหนดหรือเปลี่ยนแปลงค่าได้โดยใช้ setvar การเปลี่ยนแปลงสีที่ใช้จึงจะต้องใช้คำสั่งของ AutoCAD คือ COLOR เช่น

command: COLOR

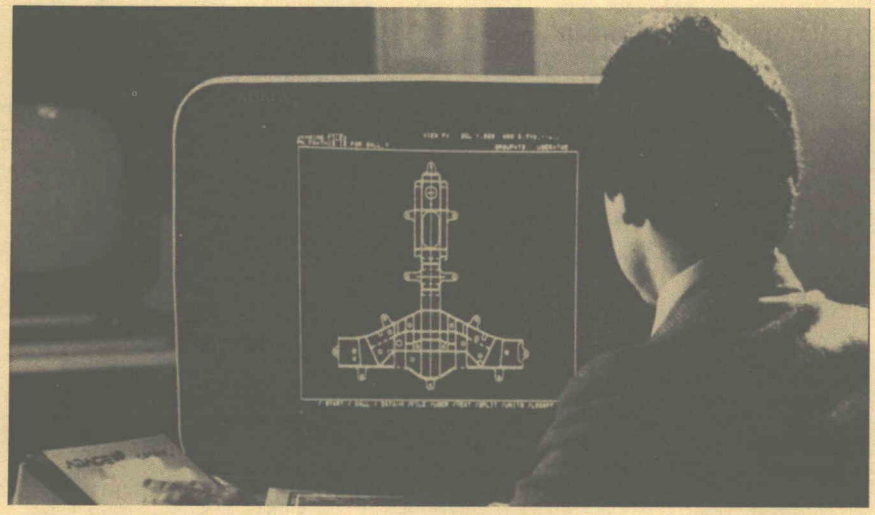
(ชื่อของสีที่ใช้เช่น RED)

ดังนั้นเมื่อเราต้องการเปลี่ยนแปลงสีโดยใช้ AutoLISP จะต้องใช้ฟังก์ชัน command เช่น

(command "COLOR" 1)

```
(defun setcolor (c)
  (if c
    (cond ((or (= c "BYLAYER") (= c "BYBLOCK"))
           (command "color" c))
          ((= c "") (command "color" "bylayer"))
          ((= (type c) 'INT) (command "color" c))
          (t (command "color" (atoi c)))
    )
    (command "color" "bylayer")
  )
)
```

ตัวอย่างที่ 2



เป็นการเปลี่ยนสีเป็นสีเบอร์ 1 ซึ่งเป็นสีแดง (ตามค่า default ซึ่งเบอร์ของสีนี้จะสามารถที่จะเปลี่ยนแปลงได้ใน CONFIGURATION MENU) ในตัวอย่างของฟังก์ชัน setcolor นี้จะมีอาร์กิวเมนต์หรือพารามิเตอร์ผ่านเข้ามาในฟังก์ชันเพียงตัวเดียวคือ c ซึ่งในที่นี้เราจะสมมติว่ามีค่าเป็น string หรือ integer เท่านั้น เพราะจุดประสงค์หลักของฟังก์ชันนี้เพื่อเรียกใช้จากฟังก์ชันอื่น ๆ ส่วนตัวอย่างที่ 3 จะคล้ายคลึงกันกับตัวอย่างที่ 2 ในหลักการอยู่แล้วจะไม่ขออธิบาย

ตัวอย่างที่ 4 จะเป็นการแสดงให้เราเห็นได้ชัดเจนมาก ถึงความแตกต่างระหว่างภาษา LISP กับภาษาคอมพิวเตอร์อื่น ๆ เราใช้ความสามารถที่ built-in (ที่ผู้สร้างภาษา LISP ให้มาโดยเฉพาะ) ในการจัดการกับลิสต์ ภาษา LISP จะช่วยให้เราเขียนโปรแกรมได้สั้นและได้ใจความมาก ซึ่งถ้าผู้ที่มีความเคยชินกับภาษาอื่น ๆ มาอ่านโปรแกรมนี้ครั้งแรกอาจจะรู้สึกงงและไม่เข้าใจ แต่ถ้าคุ้นกับฟังก์ชันของ LISP สักเพียงเล็กน้อยก็จะเข้าใจถึงทั้งการทำงานและแนวความคิดของฟังก์ชันได้ทันที ขอยกตัวอย่างการทำงานของฟังก์ชันนี้เลย เช่น

(pushvars '("BLIPMODE" "ORTHOMODE" "HIGHLIGHT"))

เช่นเดียวกันกับตัวอย่างแรกอาร์กิวเมนต์ของฟังก์ชันเป็นลิสต์ คือ ("BLIPMODE" "ORTHOMODE" "HIGHLIGHT") สิ่งที่ฟังก์ชันนี้ทำก็คือจะเก็บเอาค่าของตัวแปรระบบตามตัวอย่างข้างบน คือ BLIPMODE, ORTHOMODE และ HIGHLIGHT นี้เอาไว้ในตัวแปรที่มีชื่อว่า __vars และถ้าสมมติว่าค่าของตัวแปรนี้เป็น 1 หมด ภายหลังจากการเรียกใช้ฟังก์ชันตามตัวอย่างข้างต้นแล้วค่าของ __vars จะเท่ากับ (("BLIPMODE" 1) ("ORTHOMODE" 1) ("HIGHLIGHT" 1)) ซึ่ง __vars นี้จะเป็นลิสต์รวมของลิสต์ย่อย ๆ ที่ประกอบด้วยชื่อของตัวแปรระบบกับค่าของมัน เราสามารถตรวจสอบได้ที่ command: โดย

command: !__vars

<AutoCAD จะตอบมาว่า>

(("BLIPMODE" 1) ("ORTHOMODE" 1) ("HIGHLIGHT" 1))

สำหรับประโยชน์ของการเก็บค่าของตัวแปรระบบนี้เอาไว้ก็คือ เมื่อเราเขียนโปรแกรมบางอย่างขึ้นเพื่อใช้งาน เราต้องการที่จะให้สถานภาพต่าง ๆ ใน drawing editor หลังจากที่เราเรียกใช้โปรแกรมนั้นแล้ว เช่น orthomode, snapmode เป็นต้น ให้ยังคงอยู่เหมือนเดิม เราจะต้องเก็บค่าของตัวแปรระบบที่เราเปลี่ยนแปลงไปในโปรแกรมเอาไว้ก่อนเมื่อตอนแรกเริ่มของโปรแกรม จากนั้นเมื่อจะสิ้นสุดการทำงานของโปรแกรม

```
(defun setltype (l)
   (if l (command "linetype" "s" l ""))
)
```

ตัวอย่างที่ 3

```
; (pushvars '(<variable name> <variable name>.... ))
; Save system variables in global variable named _VARS.
; The read-only variables may be used.
(defun PUSHVARS (a)
   (setq _vars (append
                  (mapcar 'list
                     (mapcar 'strcase a)
                     (mapcar 'getvar a))
                  _vars))
)
```

ตัวอย่างที่ 4

เราก็เซตค่าต่าง ๆ นี้กลับคืนตามเดิม ซึ่งการเซตค่าต่าง ๆ นี้เป็นการทำตรงกันข้ามกับฟังก์ชัน pushvars นั่นคือ ตัวอย่างที่ 5 popvars นั่นเอง แต่ก่อนที่เราจะไปถึงตัวอย่างที่ 5 เราลองมาดูการทำงานของ pushvars กันก่อนดีกว่านะครับ

หัวใจของการทำงานของฟังก์ชัน pushvars นี้คือ append และ mapcar ขอทบทวนความจำกันก่อนสักเล็กน้อย ว่า mapcar ก็คือ ฟังก์ชันที่จะใช้จัดการกับลิสต์ให้เป็นไปตามความต้องการ เช่น จากตัวอย่าง (mapcar 'strcase a) เนื่องจากเรากำหนดให้ a เป็นลิสต์ ดังนั้นฟังก์ชัน strcase ซึ่งโดยปกติแล้วเราจะใช้กับ atom ที่เป็น string ไม่ใช่ลิสต์ ดังนั้นเราจึงต้องใช้ mapcar เข้าช่วย mapcar ตัวแรกจึงเป็นการเปลี่ยน element ของลิสต์ให้เป็นตัวหนังสือตัวใหญ่ทั้งหมด ส่วน mapcar ตัวที่สองจะเป็นการเรียกเอาค่าของตัวแปรระบบของ element นั้นออกมา เมื่อ mapcar ทั้งสองทำงานได้เต็มที่คือจนทุก element ของลิสต์เรียบร้อยเสร็จสิ้นกระบวนความแล้ว ขั้นสุดท้าย append จะทำหน้าที่รวบรวมเอาผลที่ได้จาก mapcar นั้นเป็นลิสต์เพียงอันเดียวซึ่งจะถูกเก็บเอาไว้ใน ___vars โดยฟังก์ชัน setq การเขียนโปรแกรมในลักษณะนี้จะทำให้เราเห็นได้อย่างชัดเจนว่าภาษา LISP ได้ถูกออกแบบขึ้นเพื่อเอาไว้จัดการกับข้อมูลที่เป็นลิสต์อย่างแท้จริง

ตัวอย่างที่ 5 อันที่จริงตัวอย่างนี้ก็ไม่มีอะไรยาก เพียงแต่จะเรียกใช้ฟังก์ชัน setcolor และ setltype ที่เราได้เขียนไปแล้วในตัวอย่างที่ 2 และ 3 ตามลำดับ เนื่องจากตัวแปรระบบที่เราเก็บเอาไว้โดยใช้ฟังก์ชัน pushvars อาจจะมีตัวใดตัวหนึ่งเป็นสีหรือเป็นชนิดเส้น ซึ่งสองตัวนี้เป็น Read-Only ตัวแปรที่เราไม่อาจจะเซตได้โดยใช้ setvar ตามที่ได้กล่าวไว้แล้วในตัวอย่างที่ 2 นั้น (อันที่จริงมีตัวแปรระบบตัวอื่น ๆ ที่เป็น Read-Only อีกมาก แต่เนื่องจากนี่เป็นเพียงการยกตัวอย่างเท่านั้น และตัวแปรทั้งสองตัวนี้จะถูกนำมาใช้บ่อยครั้งกว่าตัวอื่นจึงได้นำมาใส่เอาไว้ในฟังก์ชัน ดังนั้นถ้าท่านทดลองใช้ฟังก์ชันนี้กับ Read-Only ตัวแปรตัวอื่น ๆ นอกจากสองตัวนี้จะทำให้โปรแกรมเกิด error ขึ้นมา) เราจะต้องตรวจสอบดูว่าเป็นตัวแปรที่มีชื่อว่า CECOLOR (มาจาก Current Entity Color) หรือ CELTYPE (Current Entity Line Type) หรือไม่ จากนั้นจึงเรียกใช้ฟังก์ชันที่เหมาะสมต่อไป ตามที่ได้กล่าวไปแล้วว่าเมื่อเราต้องการจัดการกับลิสต์ (เนื่องจากเราเก็บเอาค่าทั้งหมดไว้ในลิสต์ที่มีชื่อว่า ___vars) ต้องใช้ mapcar และเนื่องจากไม่มีฟังก์ชันใดที่จะทำการตรวจสอบอะไร ๆ หรือทำได้อย่างใจ เราจึงจะต้องนิยามฟังก์ชันขึ้นมาใหม่ (ขอให้เปรียบเทียบดูจากตัวอย่างที่แล้วที่มีการใช้ mapcar กับ strcase และ setvar ซึ่งเป็นฟังก์ชันภายใน AutoLISP) การนิยามฟังก์ชันขึ้นมาภายใน mapcar นี้จะต้องใช้ฟังก์ชันพิเศษที่มีชื่อว่า lambda (ขอให้ดูรายละเอียดและตัวอย่างเรื่อง lambda ได้ในเรื่องฟังก์ชัน Anonymous ในแอดวานซ์ AutoCAD ในไมโครฯเล่มที่ 35 เดือนมกราคม 2531) ถ้าท่านต้องการทดสอบฟังก์ชัน popvars นี้ อย่าลืมว่าจะต้องพิมพ์หรือ load ฟังก์ชัน setcolor และ setltype เข้าไปก่อน หรือจะใช้วิธีพิมพ์เอาไว้ในไฟล์ ACAD. LISP ก็ได้ มิฉะนั้นจะเกิด error ขึ้นมาได้ และก็อย่าลืม load และเรียกใช้ pushvars ก่อนตามตัวอย่างในคำอธิบายฟังก์ชัน pushvars เสียก่อน

```
; (popvars)
; Restores the system variables in _VARS.
(defun POPVARS ()
   (if _vars (progn
      (mapcar '(lambda (x)
               (cond ((= (car x) "CECOLOR")
                        (setcolor (cadr x)))
                     ((= (car x) "CELTYPE")
                        (setltype (cadr x)))
                     (t (setvar (car x) (cadr x)))
               ))
               _vars)
      (setq _vars nil) )
   )
   (terpri)
)
```

ตัวอย่างที่ 5

รวม input เป็นลิสต์

ตัวอย่างที่ 6 เป็นฟังก์ชันที่มีประโยชน์มากอันหนึ่ง เราจะใช้ฟังก์ชันนี้เมื่อเราต้องการรวบรวมเอาคำตอบจากผู้ใช้มาเป็นลิสต์ (แน่นอนครับว่า เมื่อเราสามารถที่จะจัดอะไรต่อมิอะไรเข้าอยู่ในรูปของลิสต์ได้แล้วละก้อ ทุกอย่างคงจะ

```
;********************** Form a list of strings
(defun formlist (s e / a)
  (cond ((= e 'str) (setq a (getstring 't s)))
        ((= e 'list) (setq a (read (strcat "(" (getstring 't s) ")"))))
        ((= e 'int) (setq a (getint s)))
        ((= e 'real) (setq a (getreal s)))
        ((= e 'sym)
          (setq a (getstring 't s))
          (if (/= a "") (setq a (read a)))
        )
        ((null e) (setq a (getpoint s)))
  )
  (if (listp a)
    (if a (cons a (formlist s e)))
    (if a
      (if (/= a "")
        (cons a (formlist s e))
      )
    )
  )
)
```

ตัวอย่างที่ 6

หวานหอ- มอสระอูหมูขึ้นเยอะเลย ฟังก์ชันอันนี้ก็เอาไว้สำหรับเตรียมหมูเอาไว้ให้พร้อมแหละครับ) สมมติว่าเราจะต้องการรวบรวมจุดต่าง ๆ สักจำนวนหนึ่งให้อยู่ในรูปของลิสต์และเราต้องการให้ขึ้นคำที่ข้างล่างในช่อง command prompt ว่า Enter point: (เพื่อว่าเราจะได้ทราบว่าโปรแกรมของเราทำอะไร) จะเรียกใช้ฟังก์ชัน formlist ได้ดังนี้

(formlist "\nEnter point:" nil)

ซึ่งจะพบว่า (หลังจากที่ใส่ฟังก์ชันนี้เข้าไปใน AutoCAD ด้วยวิธี load หรืออะไรก็ตามแต่...อย่าลืมว่าถ้าได้ error ว่า Insufficient node space แล้วละก้อควรที่จะเซตค่า LISPHEAP ให้สูงสักหน่อย ดูเรื่องนี้ในแอดวานซ์ AutoCAD ตอน 4 ในไมโครฯเล่มที่ 33) ที่ช่อง command editor จะขึ้นว่า Enter point: ให้เราใส่จุดต่าง ๆ เข้าไปทีละจุดและจะขึ้นว่า Enter แล้ว ที่ช่องเดียวกันนี้จะปรากฏลิสต์ขึ้นมาอันหนึ่ง เป็นลิสต์ของจุดต่าง ๆ ตามลำดับที่คุณเพิ่งจะใส่เข้ามานั่นเอง แต่ถ้าระหว่างที่คุณใส่จุดต่าง ๆ เข้าไปนั้น ฟังก์ชันเกิดหยุดทำงานแล้วให้ error ทำนองว่า Stack overflow ขึ้นมา ขอให้ทราบไว้ว่าได้กำหนดค่าของ LISPSTACK เอาไว้ไม่เพียงพอ จึงจะต้องไปแก้ไขค่าของตัวแปรตัวนี้ใหม่ก่อนเรียกโปรแกรม AutoCAD และโปรดอย่าลืมว่าค่าของ LISPHEAP และ LISPSTACK เมื่อรวมกันแล้วจะมีค่าเกิน 45000 ไม่ได้ ดังนั้นจะต้องพิจารณาดูให้ดี ซึ่งในทางปฏิบัติถ้าเราจะใช้ฟังก์ชันประเภท recursion มาก ๆ เช่น mapcar หรือ formlist และรวมไปถึง ml ท้ายบทด้วยละก้อ จะต้องเซตค่า LISPSTACK เอาไว้อย่าให้น้อยนัก คือไม่ควรจะมีค่าต่ำกว่า 10000 (ค่าที่ทดลองว่าเหมาะสมจะอยู่ในระหว่าง 10000 ถึง 15000) และไม่ควรจะสูงเกินไปเพราะจะไปกินเนื้อที่ของ LISPHEAP ซึ่งเป็นเนื้อที่สำหรับเก็บค่าของตัวแปรและนิยามของฟังก์ชันต่าง ๆ บางท่านที่ไม่ได้ติดตามเรื่องของ AutoCAD และไม่ทราบว่าผมพูดถึงเรื่องอะไรขอให้ใส่สองบรรทัดข้างล่างไว้ใน Autoexec.bat แล้วกดบูตเครื่องใหม่ครับผม

```
set LISPHEAP = 30000
set LISPSTACK = 15000
```

ขอกลับมาเข้าเรื่องของเราต่อนะครับ อาร์กิวเมนต์ตัวสุดท้ายของตัวอย่าง formlist ข้างต้น คือ nil นั้น แสดงว่าเราต้องการรวบรวมลิสต์ของจุดต่าง ๆ สำหรับอาร์กิวเมนต์ของฟังก์ชัน formlist นี้จะเป็นค่าต่าง ๆ ได้รูปที่ 1

'str	จะรับค่าอินพุตเป็น string เช่น "A" "UTOCAD"
'list	จะรับค่าอินพุตเป็น list เช่น (2 3 4) (5 9 6)
int	จะรับค่าอินพุตเป็น integer เช่น 2 3 7
'real	จะรับค่าอินพุตเป็น real number เช่น 2.3 4.597
'sym	จะรับค่าอินพุตเป็น symbol เช่น JOHN JACK ADAM
nil	จะรับค่าอินพุตเป็น จุด (ซึ่งก็คือลิสต์นั่นเอง) เช่น (2.54) (3.05 2.10)

รูปที่ 1

ซึ่งจะเห็นว่าเราสามารถใช้ฟังก์ชันนี้เพื่อรับเอาข้อมูลต่าง ๆ เกือบทุก data type ที่ AutoLISP มี ดังนั้นฟังก์ชัน formlist นี้จะเป็นฟังก์ชันอเนกประสงค์ที่จะเป็นประโยชน์ในการเขียนโปรแกรม AutoLISP ต่อไปข้างหน้ามาก และเราจะได้พบบ่อยขึ้น

สำหรับการเขียนฟังก์ชันนี้ก็ใช้หลัก recursion โดยการเรียกใช้ตัวเองเรื่อยไปจนกว่าอินพุตจะเป็น nil การทำงานของ formlist นี้เริ่มจากฟังก์ชัน cond เป็นตัวแยก (เหมือนกับ case ในภาษาอื่น ๆ) ว่าพารามิเตอร์ ที่ผ่านสู่ฟังก์ชันตัวที่สอง คือ e นั้นมีค่าเป็นอย่างไรก็จะเลือกค่าอินพุตเป็นไปตาม data type นั้น เช่น ใช้ getint กับอินพุตที่เป็นตัวเลขจำนวนเต็ม, ใช้ getstring กับอินพุตที่เป็นตัวหนังสือ นอกจากนี้เรายังสามารถที่ใช้ฟังก์ชัน read เพื่อรับเอาอินพุตประเภทสตริงแล้วแปรเป็นสัญลักษณ์ เช่น A B หรือชื่อของตัวแปรต่าง ๆ ได้อีกด้วย ส่วนตัวอย่างการใช้งานขอให้ดูได้จากตัวอย่างที่ 7 ซึ่งจะได้อธิบายต่อไป

ML.LSP

ตัวอย่างที่ 7 เป็นตัวอย่างที่ได้นำมาจากการใช้งานจริงในงานเขียนแบบสถาปัตยกรรม ดังนั้นการออกแบบโปรแกรมจะต้องคำนึงถึงความสะดวกในการใช้งานและประสิทธิภาพของการทำงานเป็นอันดับแรก ในตอนแรกนี้จะขออธิบายถึงวิธีการเรียกใช้โปรแกรมนี้ก่อน เพื่อผู้ที่ต้องการที่จะนำโปรแกรมไปใช้งานจะสามารถนำไปใช้งานได้ทันที การเรียกใช้โปรแกรม ml. lsp นี้ จะต้องมีการเตรียมการล่วงหน้าโดย load โปรแกรมตามตัวอย่างที่ 1 ถึง 6 ที่ command: ของ AutoCAD หรือจะใช้วิธีรวบรวมตัวอย่าง 1 ถึง 6 นี้เอาไว้ในไฟล์

ตัวอย่างที่ 7

```
; ml.lsp
;   Draw wall lines in plan.
;
; (ccr <list>)
; Return list after take the last one out
(defun ccr (l)
  (reverse (cdr (reverse l)))
)

; Calculate left points
(defun list0 (p q)
  (mapcar '(lambda (x y)
    (list
      (polar x (- (angle x y) d90) d1)
      (polar y (- (angle x y) d90) d1)
    )
    ) p q
  )
)
; Calculate right points
(defun list1 (p q)
  (mapcar '(lambda (x y)
    (list
      (polar x (+ (angle x y) d90) d2)
      (polar y (+ (angle x y) d90) d2)
    )
    ) p q
  )
)
; Calculate intersections
(defun listi (p q)
   (mapcar '(lambda (x y)
            (inters (car x) (cadr x) (car y) (cadr y) nil)
            ) p q
   )
)
```

ตัวอย่าง 7 (ต่อ)

```
; Draw a wall line
(defun wall (p / p0 p1 q)
  (setq p0 (ccr p) p1 (cdr p))
  (if (null p0) (setq p0 (list (last p)) p1 p0))
  (setq q (listi p0 p1))
  (if (equal q '(nil)) (setq q nil))
  (setq q (cons (caar p0) q)
        q (append q (cdr (last p1)))
  )
  (mapcar '(lambda (x y)
    (if y (command "line" x y ""))
    ) (ccr q) (cdr q)
  )
)
(defun C:ML (/ a a0 a1 l0 l1 d1 d2)
  (pushvars '("blipmode" "osmode" "highlight"))
  (setqv '("highlight" 0 "blipmode" 1 "cmdecho" 0))
  (if (null _w) (setq _w 0.1))
  (setq lc (getstring "\nLocation L/C/R <C>: "))
  (if (= lc "") (setq lc "C"))
  (setq lc (strcase lc) d90 (/ pi 2))
  (cond ((= lc "L") (setq d1 _w d2 0))
        ((= lc "C") (setq d1 (* _w 0.5) d2 d1))
        ((= lc "R") (setq d1 0 d2 _w))
  )
  (setq a (formlist "\nEnter point: " nil)
        l0 (ccr a) l1 (cdr a)
        a0 (list0 l0 l1) a1 (list1 l0 l1))
  (setqv '("blipmode" 0 "osmode" 0))
  (wall a0)
  (wall a1)
  (popvars)
)
```

เดียวแล้วใช้ชื่อว่า ACAD. LSP จากนั้นรันโปรแกรม AutoCAD เพื่อให้ Auto CAD ทำหน้าที่ load ฟังก์ชันเหล่านี้เข้าไปใน memory เอง และไม่ว่าคุณจะเลือกใช้วิธีใดก็ตามจากนั้นที่ command: ให้พิมพ์ว่า (load "ml") (จะต้องมีไฟล์ ml. lsp อยู่ในไดเรคทอรีที่ AutoCAD สามารถเรียกมาได้ คือ default directory หรือที่ตั้งเอาไว้โดย set ACAD = หรือคำสั่ง append ของ DOS) หลังจากที่ AutoCAD ตอบคุณว่า C:ML แล้วเป็นอันว่าเสร็จพิธี ที่มี C: ขึ้นต้นหน้าชื่อฟังก์ชันนั้นหมายถึงว่า เราสามารถเรียกใช้ฟังก์ชันนี้ที่ command: โดยการเรียกชื่อฟังก์ชันโดยไม่ต้องใช้วงเล็บเปิดและปิดล้อมรอบเสมือนหนึ่งว่าเป็นคำสั่งปกติของ AutoCAD (และแน่นอนครับว่าท่านอาจจะไม่ชอบชื่อ ML ที่ผมตั้งขึ้นและอยากจะเปลี่ยนเป็นชื่ออื่นเช่น WALL ก็สามารถทำได้โดยการเปลี่ยนคำ ml ตรง (defun C:ml เป็น (defun C:WALL ได้เช่นกัน ฟังก์ชัน ml นี้เราสามารถเรียกใช้ได้ตลอดเวลาจนกว่าเราจะออกจาก AutoCAD ไปซึ่งเราจะสามารถเรียกใช้ได้ที่ command: ว่า

command: ML (กด enter)

และตอบคำถาม Location L/C/R ได้ 3 ตัวคือ

L คือ เส้นผนังด้านซ้ายมือจะขึ้นชิดเส้นร่างที่วาดไป

C คือ เส้นผนังจะขึ้นทั้งสองด้านของเส้นร่าง

R คือ เส้นผนังด้านขวามือจะขึ้นชิดเส้นร่าง

หรือเพียงแต่กด enter ก็จะเลือกเป็น C คือเส้นผนังจะขึ้นทั้งสองด้านของเส้นที่ลาก จากนั้นก็ใส่จุดไปเรื่อย ๆ ตามต้องการ เมื่อเพียงพอแล้วก็กด enter เพื่อหยุดคำถาม Enter point: โปรแกรม ml นี้จะทำหน้าที่วาดเส้นผนังตามจุดที่ใส่

เข้าไปและ trim ตามขอบและมุมให้เรียบร้อย นอกจากนี้ถ้าท่านพอที่จะทำเมนูใน AutoCAD เป็น ไม่ว่าจะเป็นบนดิจิไตเซอร์ หรือ screen menu สำหรับเมาส์ก็ดีท่านสามารถที่จะบรรจุ option ต่าง ๆ ของ ml คือ L, C, R โดยเพียงแต่กดเพียงปุ่มเดียว จากนั้นก็เลือกผนังชนิดที่ต้องการได้ ดังรูปที่ 2

การใส่จุดต่าง ๆ เข้าไปก่อนวาดผนังทำให้เกิดความรวดเร็วแก่ผู้เขียนโดยที่ไม่ต้องนั่งรอการคำนวณจุดและการ trim ขอบทีละจุด โดยยึดหลักว่าให้ผู้ใช้ใส่จุดที่ต้องการเข้าไปให้หมดเสียก่อน จากนั้นค่อยให้ AutoCAD วาดเสียทีเดียว เป็นการ minimize เวลาของผู้ใช้ให้ลดน้อยลงและให้ AutoCAD ทำงานให้เรามากขึ้น ตามความมุ่งหมายในการเขียนฟังก์ชันขึ้นเพื่อช่วยในการทำงาน การทำเช่นนี้ได้เราจะต้องเก็บรวบรวมค่าต่าง ๆ ที่ได้จากการอินพุตในแต่ละครั้งให้อยู่ในรูปของลิสต์จากนั้นก็นำลิสต์ที่ได้นี้ไปดำเนินการต่ออีกทีหนึ่ง สิ่งที่ผมอยากจะเน้นก็คือ **เราจะต้องมองว่าวิธีการแก้ปัญหามิได้มีเพียงวิธีเดียว ถ้าเรามองปัญหาตามแบบเดิมเราจะไม่สามารถที่จะหาทางออกใหม่ ๆ ที่อาจจะดีกว่าได้เลย**

ปฏิกิริยาระหว่างจุดสองจุดกลายเป็นเส้น ?

การลากเส้นเชื่อมต่อระหว่างจุดต่าง ๆ สมมติว่าสัก 4 จุดเช่น A B C D โดยปกติเราจะต้องลากเส้นจาก A ผ่าน B, C แล้วไปจบที่ D แต่ถ้าหากเราจัดจุดต่าง ๆ นี้เป็นลิสต์คือ (A B C D) แล้วเราจะสามารถลากเส้นผ่านจุดต่าง ๆ ซึ่งเป็นสมาชิกในลิสต์ได้อย่างไร? การลากเส้นถ้าจะกล่าวให้ชัดเจนก็คือการเชื่อมต่อระหว่างจุดสองจุด และถ้าเรามองว่าการลากเส้นเป็นการกระทำหรือกิริยาหรืออีกนัยหนึ่งคือกระบวนการ (procedure) ซึ่งนำเอาจุดสองจุดมาทำปฏิกิริยากันเกิดเป็นเส้นตรง ดังนี้แล้วเราจะได้ผลดังรูปที่ 3 รวมเป็นเส้นตรงสามเส้นลากเชื่อมระหว่างจุดทั้งหมดสี่จุด ซึ่งถ้าเรามองจุดแรกของทั้งสามบรรทัดข้างบนเป็นชุดจำนวนลิสต์ และจุดสองเป็นชุดจำนวนลิสต์เช่นกัน จะได้ดังนี้

(A B C)

และ (B C D)

ถ้าเราลากเส้นตรงเชื่อมจุดสองจุดซึ่งคือสมาชิกในลิสต์ข้างบนทั้งสองอันตัวต่อตัว คือตัวแรกของลิสต์แรก (คือ A) กับตัวแรกของลิสต์สอง (คือ B) จะได้เส้น A--B และตัวที่สองของลิสต์แรก (คือ B) กับตัวที่สองของลิสต์สอง (คือ C) จะได้เส้น B--C และถ้าทำดังนี้เรื่อยไปเราก็จะได้เส้นตรงเชื่อมจุดครบทั้งสี่จุดตามต้องการ ถ้าเขียนในรูปของลิสต์จะได้ว่า

(A B C)
| | |
(B C D)

จะเห็นว่าเราสามารถทำอย่างข้างบนได้ก็ต่อเมื่อเรานำเอาลิสต์ดั้งเดิมคือ (A B C D) มาแยกเป็นสองลิสต์ ซึ่งถ้าเราพิจารณาดูให้ดี ๆ แล้วเราสามารถที่จะทำได้โดยไม่ยากเลย เนื่องจากลิสต์ตัวแรกคือ (A B C) ได้มาจากการที่เอาตัว D ซึ่งเป็นสมาชิกตัวสุดท้ายออกจาก (A B C D) และลิสต์ตัวที่สองคือ (B C D) มาจากการเอาตัว A ซึ่งเป็นสมาชิกตัวแรกออกจาก (A B C D) นั่นเอง มาถึงตอนนี้หลายคนที่ติดตามเรื่องนี้มาตั้งแต่ต้นหรือผู้ที่เคยเขียนฟังก์ชัน LISP มาบ้างแล้วคงจะร้องอ๋อ!

การเอาตัวสุดท้ายออกจากลิสต์นั้นไม่มีฟังก์ชันใน AutoLISP ที่จะกระทำได้ แต่เราสามารถเขียนฟังก์ชันขึ้นได้โดย

```
(defun ccr (l)
        (reverse (cdr (reverse l)))
)
```

ขอให้ดูวงเล็บข้างในสุดและออกมาทีละวงเล็บ

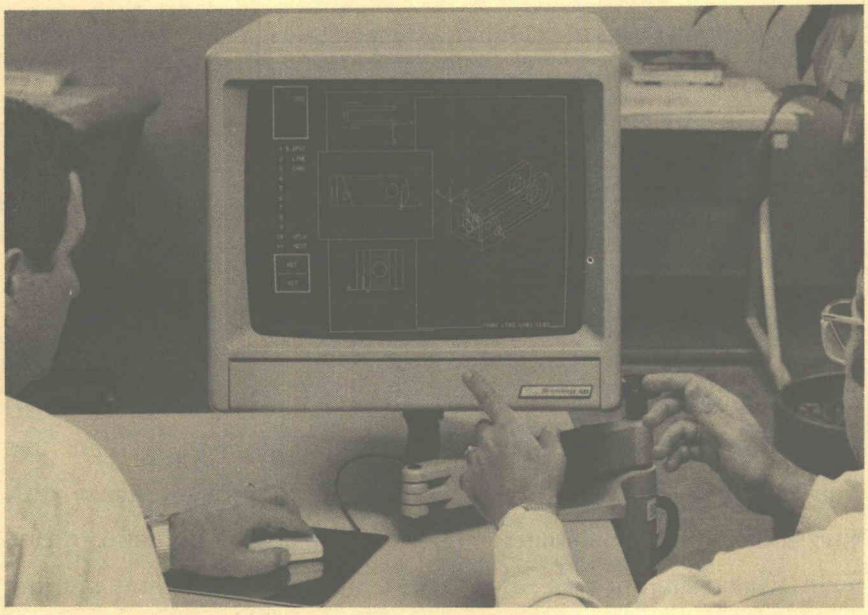
(reverse l) ในที่นี้ l คือลิสต์ที่จะผ่านเข้าสู่ฟังก์ชันคือ (A B C D)
(reverse (A B C D)) จะได้ว่า (D C B A)

รูปที่ 2

```
(WALL L)^C ^C ^P(if (null C:ML) (load "ml")) ML L
(WALL C)^C ^C ^P(if (null C:ML) (load "ml")) ML C
(WALL R)^C ^C ^P(if (null C:ML) (load "ml")) ML R
```

รูปที่ 3

จุด A และ B มาทำปฏิกิริยากันเกิดเป็นเส้นตรง A – – B
จุด B และ C มาทำปฏิกิริยากันเกิดเป็นเส้นตรง B – – C
จุด C และ D มาทำปฏิกิริยากันเกิดเป็นเส้นตรง C – – D



สำหรับลิสต์ตัวที่สองนั้นได้มาจากการเอาตัวแรกออกโดยใช้ cdr คือ (cdr (A B C D)) จะได้ลิสต์ตามที่ต้องการคือ (B C D)

กล่าวโดยสรุปแล้วการที่เราต้องการที่จะลากเส้นผ่านจุดที่เป็นสมาชิกในลิสต์นั้น เราจะต้องแยกลิสต์นั้นออกเป็นสองอัน จากนั้นจึงนำลิสต์ทั้งสองอันมาทำปฏิกิริยากันโดยการลากเส้น ซึ่งการนำเอาลิสต์มาทำอะไรกันนี้เราใช้ฟังก์ชันชื่อ mapcar และ lambda เข้าช่วยดังที่ได้เคยอธิบายไปแล้วในเรื่องนั้น ๆ

ภายในโปรแกรม ml. lsp ก็ยึดหลักของ concept ที่ได้อธิบายไปแล้วข้างต้น แต่ว่าการทำงานจะมีความสลับซับซ้อนกว่า เนื่องจากเราต้องการนำโปรแกรมที่เขียนขึ้นไปใช้งานจริง ซึ่งจะต้องวาดเส้นผนังที่เป็นเส้นตรงขนานกันสองเส้น โดยที่หนึ่งในสองเส้นนั้นอาจจะเป็นเส้นเดียวกันกับเส้นร่างหรือไม่เป็นเลยก็ได้คือออกจาก center line ดังนั้นจุดต่าง ๆ จะต้องถูกคำนวณขึ้นใหม่ รวมทั้งจุดตัดของเส้นตรงจริงเนื่องจากเราต้องการ trim ขอบของเส้นตามมุมที่ตัดกันให้เรียบร้อย นอกจากนี้เรายังต้องเก็บค่าของความกว้างของผนัง (คือ ระยะห่างระหว่างเส้นตรงที่ขนานกันสองเส้นนั้น) เอาไว้ ซึ่งเราจะเก็บค่านี้ไว้ใน __w เพราะโดยปกติเวลาเราเขียนแบบผนังมักจะมีความกว้างที่เท่า ๆ กันคือ 0.10 ม. เวลาเราเขียนเส้นผนังเราจะได้ไม่ต้องใส่ค่า __w นี้เข้าไปทุกครั้ง จนกว่าเมื่อเราต้องการที่จะวาดผนังที่มีความกว้างเป็นอย่างอื่นเช่น 0.15 หรือ 0.20 เป็นต้น เราก็สามารถที่จะเซตค่านี้ได้ใหม่โดยใช้ฟังก์ชัน setq เช่น

(setq __w 0.15)

(setq __w 0.20)

แต่ถ้าท่านพอจะเขียนเมนูใน Auto CAD ได้ ท่านสามารถที่จะใส่บรรทัดข้างต้นเอาไว้ในเมนูเลย และเนื่องจากค่าความกว้างของผนังในงานสถาปัตยกรรมหรืออื่น ๆ มักจะมีขนาดที่มาตรฐาน คือ 0.05, 0.10, 0.15, 0.20,...หรือมิฉะนั้นเราอาจจะเขียนโปรแกรมให้ถามถึงค่าที่ต้องการได้ดังรูปที่ 4

รูปที่ 4

```
(W__Width)^C ^C ^P(if (null __w) (setq __w 0.1)); +
(setq __w1 (getreal (strcat "\nEnter Wall width<" (rtos
__w 2 2) ">:")));\ +
(if (not(or (null __w1) (< = __w1 0.000001)) (progn (setq
__w __w1) (setq __w1 nil)))
```

ขอให้ดูที่บรรทัด (defun C:ML... ซึ่งเป็นการสร้างฟังก์ชันสุดท้ายสำหรับโปรแกรมนี้ จะเปรียบเสมือนจุดเริ่มต้นของโปรแกรม ml บรรทัดต่อมาเรียกใช้ pushvars ให้เก็บเอาค่าของ blipmode, osmode และ highlight จากนั้นก็เปลี่ยนแปลงค่าของ highlight เป็น 0 เพื่อมิให้เส้นที่เราเขียนไปแล้วเปลี่ยนเป็นเส้นประเมื่อเวลา select (ดังเช่นเวลาเลือกวัตถุเมื่อต้องการ edit) และ blipmode เป็น 1 เพราะเราต้องการเห็นตำแหน่ง mark สำหรับกำหนดจุดเริ่มต้นและสิ้นสุดของเส้นผนังที่เราจะวาด และ cmdecho เป็น 0 เพื่อมิให้ prompt ตามขั้นตอนต่างของ command ของการวาดหรือแก้ไขรูปปรากฏขึ้นที่ช่อง command:

บรรทัดต่อมาเป็นการ test ค่าของ __w ที่เราตกลงเอาไว้ข้างต้นแล้วว่าจะเป็นค่าของความกว้างของผนังที่จะวาด เมื่อแรกเริ่มค่านี้จะไม่ได้ตั้งเอาไว้ว่าจะเป็นอะไรคือ เป็น nil บรรทัดนี้จะเซตค่าของ __w เป็น 0.10 ม. คือเป็นค่าของความกว้างผนังที่ใช้กันอยู่โดยปกติ จากนั้นจะถามถึงตำแหน่งของผนังที่จะวาดว่าจะชิดไปกับเส้นร่างด้านใด ซ้าย, ขวา หรือให้ออกจากตรงกลาง ถ้าท่านไม่เข้าใจก็ขอให้ทดลองดูโดยเลือก L หรือ C หรือ R ดูแล้วทดลองวาดและสังเกตดูเส้นของผนังที่มีสองเส้นนั้น ว่าเส้นด้านใดอยู่ตำแหน่งเดียวกันกับแนวเส้นที่ท่านได้กำหนดขึ้น

ชุดบรรทัดของ (cond...) ต่อมาอีก 4 บรรทัด จะเป็นการคำนวณระยะขึ้น

เพื่อที่จะใช้กำหนดตำแหน่งของเส้นคู่ผนังข้างหน้า โดยให้ระยะ d1 เป็นระยะของเส้นผนังด้านที่หนึ่งกับเส้นร่าง และระยะ d2 เป็นระยะของเส้นผนังด้านที่สองกับเส้นร่างตามลำดับ

ต่อมาเป็นชุดบรรทัดของ (setq...) ดังที่ได้กล่าวไว้แล้วว่าเราสามารถใช้ setq กับการกำหนดค่าตัวแปรพร้อมกันหลายตัวได้ ตัวแรกจะเป็น a ซึ่งจะให้เท่ากับผลที่ได้จากฟังก์ชัน formlist ซึ่งเป็นการเรียกใช้ฟังก์ชันให้ถามจุดเริ่มต้นของเส้นผนังและจุดต่อไปเรื่อย ๆ จนกว่าผู้ใช้จะกด enter ซึ่งค่าของ a นี้จะเป็นลิสต์ของจุดต่าง ๆ ตามตำแหน่งที่ใส่เข้าไปนั่นเอง ส่วนเส้นร่างบอกระยะจะขึ้นตามเคอร์เซอร์ที่เราลากไปเป็นไปตามการเรียกใช้ฟังก์ชัน (getpoint เมื่อกำหนดจุดร่วมด้วย จะปรากฏเส้นร่างออกจากจุดนั้น ๆ และระยะจะขึ้นที่ข้างบนตรงตำแหน่ง coordinate (ถ้าเราให้ COORDS ON คือกด F6)

จากนั้นจะแตกลิสต์ a นี้ออกเป็นสองลิสต์คือ 10 และ 11 เพื่อที่จะใช้คำนวณจุดตั้งต้นและสิ้นสุดของเส้นตรงสำหรับฟังก์ชัน listo เพื่อใช้คำนวณเส้นผนังด้านซ้ายของเส้นร่าง (คือเส้นที่ขนานกับทิศทางการเคลื่อนที่ของเส้นตรงไปตามระยะเชิงมุมเท่ากับ 180 องศา) และฟังก์ชัน list1 เพื่อคำนวณเส้นด้านขวา ซึ่ง list1 และ list2 นี้จะให้ค่า return มาเพื่อเก็บอยู่ในค่าของ a0 และ a1 ตามลำดับ และทั้ง a1 และ a2 นี้จะเป็น list ของเส้นตรงทั้งชุดดังรูปที่ 5

รูปที่ 5

(((จุดตั้งต้นของเส้น) (จุดสิ้นสุดของเส้น) ((จุดตั้งต้นของเส้น) (จุดสิ้นสุดของเส้น))...)

เช่น

(((X1 Y1) (X2 Y2)) ((X2 Y2) (X3 Y3))...((Xn – 1 Yn – 1)(Xn Yn)))

ซึ่งการแยกเส้นผนังทั้งหมดออกเป็นเส้นตรงย่อย ๆ ซึ่งเราแทนด้วยลิสต์ของจุดเริ่มต้น (X1 Y1) และจุดสิ้นสุด (X2 Y2)...เมื่อเราได้เช่นนี้แล้ว โดยแนวความคิดที่ได้ให้เอาไว้แล้วแต่ตอนต้นว่าถ้าเราต้องการจะเอาสมาชิกของลิสต์มาทำปฏิกิริยากัน ไม่ว่าจะเป็นการลากเส้นเข้าหากันหรืออื่น ๆ เราจะต้องแยกลิสต์นั้น ๆ ออกเป็นสอง แล้วถ้าเราต้องการจุดตัดกันระหว่างเส้นตรงสองเส้นล่ะ ใน AutoLISP มีฟังก์ชันสำหรับคำนวณหาจุดตัดของเส้นตรงสองเส้นคือ inters ซึ่งมีวิธีการเรียกใช้ตามแบบ

(inters A B C D < T หรือ nil >)

A คือ จุดเริ่มต้นของเส้นตรงเส้นแรก
B คือ จุดสิ้นสุดของเส้นตรงเส้นแรก
C คือ จุดเริ่มต้นของเส้นตรงเส้นที่สอง
D คือ จุดสิ้นสุดของเส้นตรงเส้นที่สอง

ส่วน T หรือ nil นั้น เมื่อใส่ T จะหมายถึงว่าจะเป็นจุดตัดจริงของเส้นตรงทั้งสองเส้นนั้น แต่หากเป็น nil นั้นให้คิดเหมือนว่าเส้นตรงทั้งสองมีความยาวไม่จำกัด คือจะไปตัดกันที่ใดก็ได้ หรืออีกนัยหนึ่งเมื่อเราใส่ว่าเป็น T นั้นแสดงว่าจุดตัดจะต้องอยู่ในเส้นตรงทั้งสองเส้นนั้น ส่วน nil นั้นจุดตัดอาจจะอยู่บนเส้นตรงแต่เพียงเส้นเดียวหรือไม่อยู่เลยทั้งสองเส้นก็ได้

แต่สำหรับเรื่องของเรานี้ เราไม่ได้ต้องการจุดตัดของเส้นตรงเพียงสองเส้น ดังนั้นเราจึงต้องแยกลิสต์ของชุดเส้นตรงนี้ออกเป็นสองลิสต์ จากนั้นจึงนำมาทำการ inters กันระหว่างสมาชิกของลิสต์ ซึ่งก็คือเส้นตรงนั่นเอง ตามในฟังก์ชัน wall ซึ่งฟังก์ชัน inters นี้จะให้ค่ากลับมาเป็นค่าของจุดตัด ซึ่งหมายความว่าจากลิสต์ของเส้นตรงซึ่งมีสมาชิกประกอบด้วยจุดเป็นคู่นั้น จะเหลือเป็นเพียงลิสต์ของจุดเดียวคือลิสต์ของจุดตัดของเส้นตรง จากนั้นแล้วเราจะต้องสร้างลิสต์ใหม่เพราะว่าจุดต่าง ๆ ได้เปลี่ยนแปลงไปยกเว้นจุดเริ่มต้นและจุดสุดท้ายของชุดของตำแหน่งต่าง ๆ ที่ได้รับจากผู้ใช้แต่ต้น สุดท้ายจึงนำเอาลิสต์นี้ไปแตกเป็นสองอันเพื่อที่จะนำมาลากเส้นเป็นอันเสร็จสิ้นขบวนการในการลากเส้นผนังเพียงด้านเดียว ส่วนผนังอีกด้านหนึ่งก็ใช้วิธีเดียวกัน

สำหรับบางบรรทัดเช่นในฟังก์ชัน wall จะเป็นการตรวจสอบค่าเพื่อความถูกต้องในการสร้างลิสต์จะไม่อธิบายลงไปในรายละเอียดว่าแต่ละบรรทัดทำงานกันอย่างไร จึงขอให้ท่านผู้อ่านได้โปรดศึกษาการทำงานของฟังก์ชันเอาเอง ถึงแม้จะไม่ได้เขียน comment เอาไว้ เนื่องจากต้องการให้ฟังก์ชันแลดูสะอาดตา และอ่านง่ายสำหรับผู้ที่คุ้นเคยภาษา LISP ดีอยู่แล้ว แต่การเขียนภาษา AutoLISP นี้ได้พยายามเขียนไปทีละขั้นตอนไม่ได้รวบรัด ซึ่งคิดว่าคงจะพอที่จะทำความเข้าใจได้โดยไม่ยากลำบากเกินไปนัก ขอให้ผู้ที่สนใจใช้เวลาและความพยายามสักหน่อย รับรองว่าเมื่อท่านสามารถเข้าใจการทำงานของโปรแกรมเหล่านี้แล้ว ท่านจะสามารถที่จะเขียนฟังก์ชันภาษา AutoLISP ขึ้นเพื่อช่วยในการทำงาน AutoCAD อย่างคล่องแคล่วได้ในไม่ช้า แล้วผมจะนำฟังก์ชันที่มีประโยชน์และให้ความรู้เกี่ยวกับภาษา LISP ทำนองนี้มาเสนออีก

พบกันใหม่ในโอกาสหน้าครับ
...สวัสดี